2207/7085
PATENT

UNITED STATES PATENT APPLICATION
FOR

METHOD AND APPARATUS FOR PREDICTING BRANCHES USING A META PREDITOR

INVENTORS:

Stephan J. Jourdan

Adi Yoaz

Mattan Erez

Ronny Ronen

PREPARED BY:

KENYON & KENYON
1500 K ST., N.W.
WASHINGTON, D.C. 20005
(202) 220-4200

METHOD AND APPARATUS FOR PREDICTING BRANCHES USING A META PREDICTOR

Technical Field

The present invention relates to predicting branches within a processor. More particularly, the present invention relates to a method and apparatus for using a meta predictor to predict branches for a branch instruction.

Background of the Invention

State of the art microprocessors achieve high performance by processing multiple instructions per cycle and by implementing deep pipelines. To reduce latency while executing the instructions, processors implement predictors to predict if a branch may be taken by a branch instruction that is waiting on a concurrently executing instruction. Mispredictions occur when the branch prediction is incorrect. When a misprediction is detected, pipeline flushes occur to resume execution on the correct path. The pipeline flushes are a major limitation to processor performance. This limitation especially is harsh for deep and wide machines on most modern processors. The time between a branch misprediction and the resumption of execution on the correct path is wasted by processing instructions along the wrongly predicted path. Thus, processors that improve their branch prediction accuracy can reduce mispredictions and increase their overall performance by performing more work in less time.

Fig. 1 depicts a block diagram of an instruction pipeline that is known in the art. Instruction 1 is processed by pipeline 10. Instruction 2 and other subsequent instructions also are processed by pipeline 10. Thus, instructions share the same pipeline. Pipeline 10 follows a repeated order of stages for executing the instructions. The following discussion describes the stages for executing instructions on pipeline 10. Fetch 11 fetches the instruction 1 from memory. Decode 12 decodes instruction 1. For example, decode 12 may determine if instruction 1 is an add, load or branch instruction. Read 13 reads the source operand values of instruction 1. Instruction 1 is ready to be executed. Execute 14 executes instruction 1. Write 15 writes the result of execute 14 to memory or a register specified by instruction 1. Retire 16 retires the instruction 1, and frees resources.

Instruction 2 follows the same stages as instruction 1. Pipeline 10 uses fetch 11, decode 12, read 13, execute 14, write 15 and retire 16 to process instruction 2. Instruction 2 is in a stage behind instruction 1 in pipeline 10. While instruction 1 is in the decode stage, instruction 2 is in the fetch stage. If an instruction 3 is fetched, then instruction 2 is in the decode stage and instruction 1 is in the read stage. Every stage is working on a different instruction at a given time. For example, instruction 1 may be ADD EAX, EBX. This instruction will add the contents of register EBX to the contents of register EAX, and store the result in register EAX. Instruction 2 may be ADD ECX, EAX. This instruction will add the contents of register EAX to the contents of register ECX, and store the results in register ECX. Instruction pipeline 10 waits until write 15 of instruction pipeline 10 to receive the value for EAX before read 13 may be executed.

Additional concerns arise when instruction 2 is a branch. Fetch 11 fetches instruction 2, but does not know which instruction is to be fetched next. Until the condition of the branch instruction 2 is resolved, fetch 11 is stalled. Thus, if instruction 2 is BRANCH (EAX=0), GO 200, fetch 11 will not fetch any more instructions until instruction 2 is processed by execute 14. Once the condition is evaluated by the execution stage, the target of the branch is known and fetch 11 resumes. Cycles are wasted as instruction 2 is being processed until execute 14 to fetch the next instruction. Modern processors seek to reduce this latency period by predicting the direction that instruction 2 will take. As discussed above, branch predictors may be used to predict when a branch is taken.

Mispredictions occur when the wrong direction is predicted by the branch predictor. In the example above, the branch predictor for instruction 2 may predict 200 as the probable branch target, which is taken. Instruction 1, however, yields a different result because EAX does not equal, causing instruction 2 to mispredict. Instructions processed after the bad fetch of the misprediction are flushed. As a result, all the work performed processing the instructions starting at address 200 is discarded, and execution resumes with the instruction sequentially following instruction 2.

Prediction schemes exist for implementing branch predictors to reduce the penalty associated with branch mispredictions. A branch predictor speculates on whether the branch is taken or not taken. Branch predictors generally include a target address buffer to record branch target addresses and a prediction table to deliver predicted directions. A target address buffer will indicate whether the target address is a branch, and the target of the branch. The prediction table may implement a prediction scheme that facilitates an accurate prediction for the branch instruction. A taken result may be indicated by a 1, and a not taken result may be indicated by a 0.

One scheme is the "last time" method that simply stores a bit in the branch predictor for every branch instruction that indicates if the branch was taken or not taken the last time the branch was executed. If the branch was taken last time, then the prediction is to take the branch. Another scheme is the "bimodal" method that stores two bits for every branch (modulo the size of the predictor tables) in the branch predictor. Like the last time method, the bimodal method updates the bits depending upon the final direction of the branch instruction. A taken branch results in an increment of the related two-bit counter while a not-taken branch results in a decrement. Counters saturate on both ends. The upper two states lead to a taken prediction, and the lower two states to a not-taken prediction.

Another scheme is the local prediction method. The local prediction method looks at the outcomes of previous instances of the current branch. The local prediction method uses a field in the target address buffer to store bits for these last N instances of that branch. For each new prediction, the bits indicating taken/not taken results will be shifted and the new outcomes inserted. Thus, older results are moved out of the prediction field, while more recent results are stored. This method still uses a prediction table with a 1 or 2 bit scheme, as discussed above. While the bimodal scheme uses only the address of the branch instruction to index the prediction table, the local scheme uses the outcome of past instances in addition to the index.

Another scheme is the global prediction method. The global prediction method looks at the outcomes of N preceding branches. A field or register builds a history, similar to the local prediction method, but the history will be of the last N previous branches in program order. As a branch is taken or not taken, the field or register shifts to update the history. The prediction table is indexed by both the address of the branch instruction and the content of this history register. A hybrid scheme also exists that combines the local and global prediction methods. This scheme may select which method to use. Both methods are executed with the results being input to a multiplexer. A predictor predictor predicts the method that would give the best prediction.

The methods discussed above are all based on previous branch outcomes. The methods do not correlate misprediction data to improving prediction efficiency.

Brief Description of the Drawings

Fig. 1 illustrates a block diagram of an instruction pipeline.

Fig. 2 illustrates a block diagram of a meta predictor apparatus according to an embodiment of the present invention.

Fig. 3 illustrates a block diagram of a branch predictor apparatus having a meta predictor apparatus according to an embodiment of the present invention.

Fig. 4 illustrates a flowchart of a method for predicting branches according to an embodiment of the present invention.

Fig. 5 illustrates a flowchart of a method for resolving a branch misprediction within a branch predictor and a meta predictor in accordance with an embodiment of the present invention.

Detailed Description

An embodiment of the present invention is directed to a branch predicting apparatus that reduces branch mispredictions in a processor. The branch prediction apparatus includes a misprediction history register. The branch prediction apparatus includes a meta predictor that receives an index value and a branch prediction to generate a misprediction value in accordance with the misprediction history register. The branch prediction apparatus also includes a logic gate that receives the branch prediction and the misprediction value to generate a final prediction. The final prediction may be used to predict whether a branch is taken or not taken.

Referring now in detail to the drawings wherein like parts are designated by like reference numerals throughout, Fig. 2 depicts a meta predictor apparatus 100 according to an embodiment of the present invention. Meta predictor apparatus 100 includes meta predictor 104. Meta predictor 104 receives index information 106 and branch prediction 108 and reads from base misprediction history register 110 in generating misprediction value 112. Final value 118 is the final value after the instruction has been processed, and the branch resolved. If branch prediction 108 is correct, then it should equal final value 118. If meta predictor apparatus 100 predicts that branch prediction 108 is incorrect, then meta predictor 104 generates misprediction value 112 to alter branch prediction 108. This altered value is used as the predicted outcome of the branch instruction, and should be compared to final value 118 at the execution stage.

According to an embodiment of the present invention, meta predictor 104 is a misprediction predictor. Typically, branch mispredictions are not uniformly distributed. The probability of several mispredictions occurring close together is high. For example, a misprediction distribution should not resemble a uniformly distributed process or, alternatively, a memory-less random process. Instead, the probability of a small misprediction distance is large. Thus, clusters of mispredictions may be common. Misprediction clusters may be explained by the fact that current branch predictors strive to reach a stable state. A misprediction may be an indicator for possible unstable events that disrupt the regularity learned by the branch predictor. Therefore, branch predictors are likely to mispredict before attaining another stable state.

Meta predictor apparatus 100 utilizes the correlation between the outcomes of past branches, the predicted outcome, and the correctness of previous predictions. Meta predictor 104 detects incorrect predictions by correlating the correctness of the current prediction, or branch prediction 108, with the correctness of previous predictions, as determined by base misprediction history register 110. By exploiting the clustering correlation, meta predictor 104 is able to detect unstable conditions and their effects on the correctness of following predictions.

As discussed above, base misprediction history register 110 reflects the correctness of the base predictor standing alone. Unlike global history registers that record whether previous branches were taken or not taken, base misprediction history register 110 records whether previous branch predictions were correctly predicted by the base predictor. Base misprediction history register 110 may be similar to the register used in global prediction methods in that base misprediction history register 110 is concerned with what previous branches have done. The size of base misprediction history register 110 may be variable, depending on the desired amount of global misprediction history data.

The base misprediction history register 110 may be updated by comparing the final value 118 to the branch prediction 108. Base misprediction history register 110 reflects whether the last N instances of branch prediction 108 have been correct or incorrect. If branch prediction 108 does not equal final value 118, then base misprediction history register 110 inserts a 1 by shifting the register. If branch prediction 108 equals final value 118, then the original branch prediction is correct and base misprediction history register 110 inserts a 0 by shifting the register. Thus, clusters of mispredictions are stored. Further, because mispredictions tend to occur in clusters, base misprediction history register 110 should include at least one misprediction prior to carrying out any meta predictions via meta predictor 104. If base misprediction history register 110 is nothing but 0 values, then meta predictor 104 would not reverse any branch prediction 108. Meta predictor 104 should be bypassed, and not be accessed, if base misprediction history register 110 is filled with 0 values. A transition should occur before meta predictor 104 is brought into the prediction operations for branch prediction 108. Thus, in cases where base misprediction history register 110 is all 0 values, branch prediction 108 may not be reversed or altered by meta predictor 104.

Index value 106 may be indexing information referring to the branch instruction. This information may include control-flow indicators such as the branch instruction pointer. Index value 106 also may include compressed path information, the outcomes of previous branches, the outcomes of previous occurrences of the current branch, or heuristics based on the branch type or program structure. Branch prediction 108 is an output from a base predictor that indicates a branch instruction should be taken or not taken. As discussed above, the base predictor may employ any prediction scheme, including local or global prediction schemes. Both index value 106 and branch prediction 108 are input into meta predictor 104.

Meta predictor 104 also reads misprediction history data 114 from base misprediction history register 110. Using misprediction history data 114, branch prediction 108 and index value 106, meta predictor 104 generates misprediction value 112. Misprediction value 112 then may be used to decide whether to reverse the prediction provided by the base predictor, or branch prediction 108.

Meta predictor 104 may implement any scheme suitable for binary prediction. For example, meta predictor 104 can implement a two level prediction scheme, much like most branch predictors.

Meta predictor 104 includes a two (2) bit counter for every possible index value modulo the size of the tables in this embodiment. The index is formed by index value 106, branch prediction 108, and misprediction history data 114. Prediction schemes using the two bit register may be implemented to predict prediction value 112. Because of the two bit counter configuration, each index may have four possible values in meta predictor 104. These values are updated as final value 118 is determined by incrementing/decrementing the counters. If branch prediction 108 is equal to final value 118, then the base prediction was correct. The corresponding counters in meta predictor 104 for index value 106 are decremented to reflect the correctness of branch prediction 108. If branch prediction 108 is not equal to final value 118, then the base prediction was incorrect and the corresponding counters in meta predictor 104 are incremented. Such updates may be performed only if the content of the base midprediction history register 110 is not all zeros (0) prior to the branch.

Fig. 3 depicts a branch prediction apparatus 200 according to an embodiment of the present invention. Branch predictor 202 is coupled to meta predictor 104. Branch predictor 202 generates branch prediction 108 according to a prediction scheme implemented by branch predictor 202. Branch predictor 202 receives index value 106. As discussed above, index value 106 may be any information used by the prediction scheme in branch predictor 202. Prediction schemes use index value information in determining whether a branch should be taken or not taken. For example, index value 106 may be an instruction pointer address for the address of the branch instruction. Branch prediction 108 is a taken/not taken prediction generated by branch predictor 202.

Branch predictor 202 is updated with final value 118. Because branch predictor 202 is not concerned with the "correctness" of branch prediction 108, a comparison should not be made between branch prediction 108 and final value 118. The history register of branch predictor 202 is updated with the value of final value 118, while counters within branch predictor 202 are incremented/decremented if the branch was taken or not taken for the corresponding index value 106.

Meta predictor 104 receives index value 106, branch prediction 108 and reads from base misprediction history register 110. Base misprediction history register 110 reflects the correctness of branch predictor 202. By using index value 106, meta predictor 104 uses the same index information as branch predictor 202. As discussed above, meta predictor 104 may implement a prediction scheme similar to branch predictor 202. For example, meta predictor 104 may implement a two level prediction scheme using two bit saturating counters. The difference between meta predictor 104 and branch predictor 202 involves the indexing function, such as the added misprediction history register information, and the branch prediction 108 generated by branch predictor 202. The indexing function may be a concatenation of the different inputs to meta predictor 104, or a complex hash function. As with other cached structures, many different possibilities exist for distributing the indexing information from index value 106 between the actual index and the value used to tag each meta predictor entry, if there is a tagged structure. Misprediction value 112 is generated according to the current state of the two bit saturating counters associated with the accessed entry, or branch instruction.

Misprediction value 112 is used to augment branch prediction 108. Logic gate 214 receives misprediction value 112 and branch prediction 108. Logic gate 214 determines whether to reverse branch prediction 108 according to misprediction value 112. If misprediction value 112 predicts that branch prediction 108 is correct, then logic gate 214 does not reverse branch prediction 108. If misprediction value 112 predicts that branch prediction 108 in incorrect, then logic gate 214 reverses branch prediction 108. Using the inputs, logic gate 214 generates a final prediction 216. Final prediction 216 predicts whether the branch instruction should be taken or not taken.

On a branch misprediction by branch prediction apparatus 200, the contents of base misprediction history register 110 is restored to its value just after the prediction of the faulting branch. A similar process is performed for the history register of branch predictor 202.

Thus, meta predictor 104 increases the accuracy of binary predictors, such as branch predictor 202. The improved predictor performance is achieved by using the correct/incorrect prediction information within misprediction value 112, as well as the regular information used for prediction. Meta predictor 104 extends the correlating mechanisms of current predictors and works with any base predictor. The increased branch prediction accuracy translates directly into processor performance speedup by reducing the number of pipeline flushes.

Although the embodiments discussed above were in the context of branch predictors, the concept of meta prediction is not limited to the domain of branch prediction. The present invention may be extended to other binary predictors.

Fig. 4 depicts a flowchart of a method for predicting branches according to an embodiment of the present invention. At 400, a branch predictor receives an index value. At 402, the branch predictor generates a branch prediction. At 404, a meta predictor receives the index value, the branch prediction and a misprediction value from a base misprediction history register. The base misprediction history register stores the misprediction history of the branch predictor. At 406, the meta predictor generates a misprediction value. At 408, a logic gate generates a final prediction value according to the branch prediction and the misprediction value. At 410, a final value is determined by the instruction being executed and processed, and the branch being resolved. At 412, the final value is used to update the meta predictor, the base misprediction history register, and the branch predictor. By comparing the final value to the branch prediction, the meta predictor and the base misprediction history register note whether the base branch prediction was correct. Using this information, the meta predictor then may predict whether a branch prediction is correct. The branch predictor is updated by the final value with regard to whether the branch was taken or not taken. The history register of the branch predictor also is updated.

Fig. 5 depicts a flowchart of a method for resolving a branch misprediction within a branch predictor and a meta predictor in accordance with an embodiment of the present invention. At 500, a branch misprediction is detected within the instruction pipeline. Branch mispredictions occur when a branch predictor mispredicts whether a branch is to be taken or not taken. The instruction pipeline has processed subsequent instructions according to the incorrect branch prediction. At 502, the instruction pipeline is flushed of the incorrectly processed instructions, and placed at the fetch stage for the branch instruction that was mispredicted. At 504, the branch predictor history register is restored to its value previous to the mispredicted branch instruction. The branch predictor history register is restored to the last N occurrences prior to the mispredicted branch instruction. At 506, base misprediction history register is restored to the last N occurrences prior to the mispredicted branch instruction. As noted above, base misprediction history register reflects the correctness or incorrectness of recent branch instructions, and is restored to reflect this history prior to the mispredicted branch instruction.

Thus, it is apparent that there has been provided, in accordance with the embodiments of the present invention disclosed above, a method and apparatus for predicting branches using a meta predictor. Although the present invention has been disclosed in detail, it should be understood that various changes, substitutions, and alterations may be made herein. Moreover, although software and hardware are described to control certain functions, such functions can be performed using either software, hardware or a combination of software and hardware, as is well known in the art. Other examples are readily ascertainable by one skilled in the art and may be made without departing from the spirit and scope of the present invention as defined by the following claims.